沈阳化工大学（）

课程设计报告

课程名称：软件实训

院 系：计算机科学与技术

专 业：计算机科学与技术

班 级：1601

小组 成员：张晓峰 邓云 李师州 苏超辉

指导教师：张雪

提交日期： 2018 年 7 月 14 日

目 录

[一. Small C语言语法说明（标题1+三号+黑体+加粗+居中） 3](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc1011)

[二. 词法分析器设计（三号+黑体+加粗+居中） 4](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc7786)

[2.1 单词编码设计（三号+楷体+加粗） 4](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc22963)

[2.2 状态转换图（三号+楷体+加粗） 4](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc24378)

[2.3 状态转换矩阵（三号+楷体+加粗） 4](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc25133)

[2.4 数据结构设计（三号+楷体+加粗） 4](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc12716)

[2.5 函数设计（三号+楷体+加粗） 4](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc16729)

[三. 语法分析器设计（标题1+三号+黑体+加粗+居中） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc16970)

[3.1 Small C语言的文法设计（三号+楷体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc28732)

[3.2 LL1分析表（三号+楷体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc20995)

[3.2.1 计算First集（四号+宋体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc12299)

[3.2.2 计算Follow集（四号+宋体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc31051)

[3.2.3 计算Select集（四号+宋体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc27438)

[3.2.4 LL1分析表（四号+宋体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc1425)

[3.3 语法分析器数据结构设计（三号+楷体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc7027)

[3.4 语法分析器函数设计（三号+楷体+加粗） 5](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc19221)

[四. 主函数设计（三号+黑体+加粗+居中） 6](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc18184)

[五. 测试（三号+黑体+加粗+居中） 7](file:///D:\git-space\university-compiler\计算机软件实训代码及设计报告模板\计算机软件实训资料\课程设计报告模版.doc#_Toc22168)

# Small C语言语法说明（标题1+三号+黑体+加粗+居中）

给出Small C语言的语法规定内容

1. 数据类型

|  |  |  |
| --- | --- | --- |
| 类型名 | 关键字 | 支持的运算 |
| 整型 | Int | +，\* |
|  |  |  |

1. 语句
2. 变量定义  
    分别介绍定义一个变量和定义相同类型的多个变量的语法格式，并举例。

Int 变量名；（多个变量用逗号隔开），最后以分号结束

举例：int a,b,c;

Int a;

1. 赋值语句  
    语法格式：变量名=表达式；

举例：a=b+c\*d;

1. 主程序

主程序的语法格式：介绍主程序的书写格式，并举例。

类似于c语言，但是没有头文件，没有大括号，没有主函数，直接写主程序

举例：

int a,b,c,d;

b=1;

c=2;

d=3;

a=b+c;

标识符的构词规则 ：一个字母和数字的组合,数字不能开头

# 词法分析器设计（三号+黑体+加粗+居中）

简要介绍利用有限自动机设计词法分析器的大概思路。

从初始状态识别字符，转入其他状态，当转入终态时识别单词成功，重新转入初态，再次识别，直到识别所有。

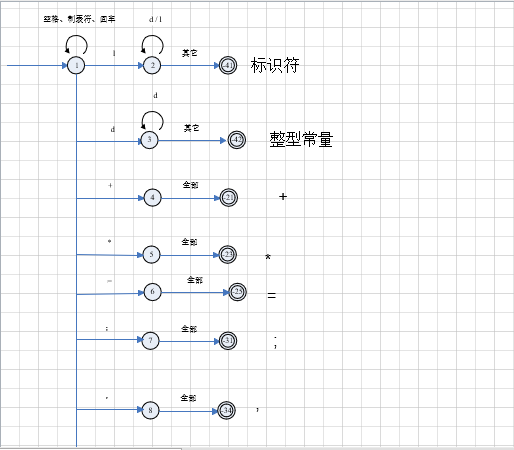
## 单词编码设计（三号+楷体+加粗）

正文。（正文采用宋体+小四+行距1.5倍）

此处请写设计文档模板中的“终结符（单词）编码表”

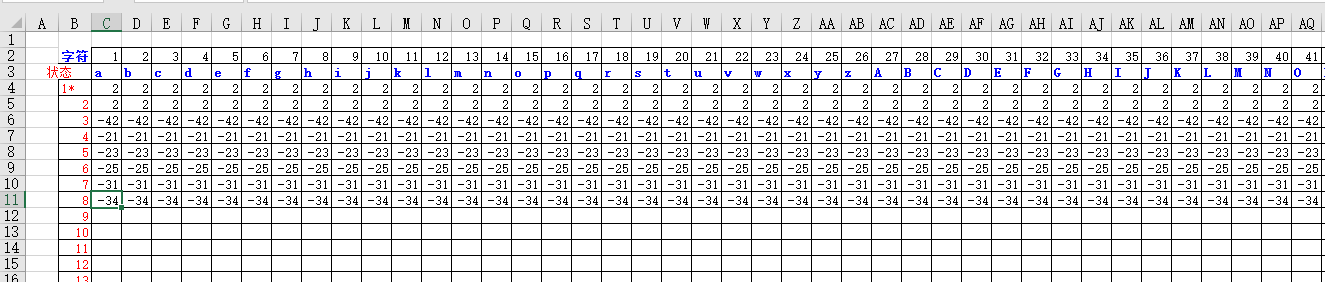
|  |  |  |
| --- | --- | --- |
| ID | 符号 | 备注 |
| 1 | 标识符 | 标识符 |
| 2 | int | 关键字 |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 | , | 界符 |
| 20 | ; |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 | + | 运算符 |
| 31 | \* |
| 32 | = |
| 33 |  |
| 34 |  |
| 35 |  |
| 36 |  |
| 37 |  |
| 38 |  |
| 39 |  |
| 40 |  |
| 41 |  |
| 42 |  |
| 43 |  |
| 44 |  |
| 45 | 整型常量 | 常量 |
| 46 |  |
| 47 |  |
| 48 |  |

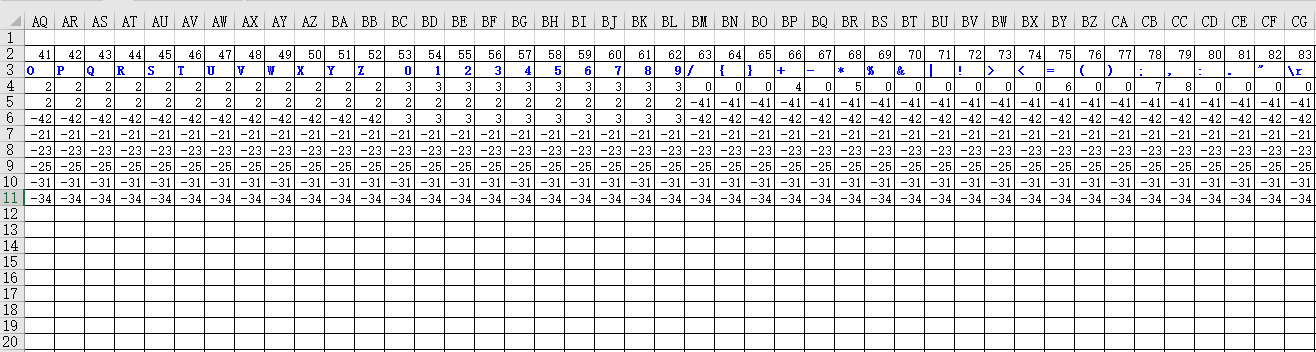
## 状态转换图（三号+楷体+加粗）

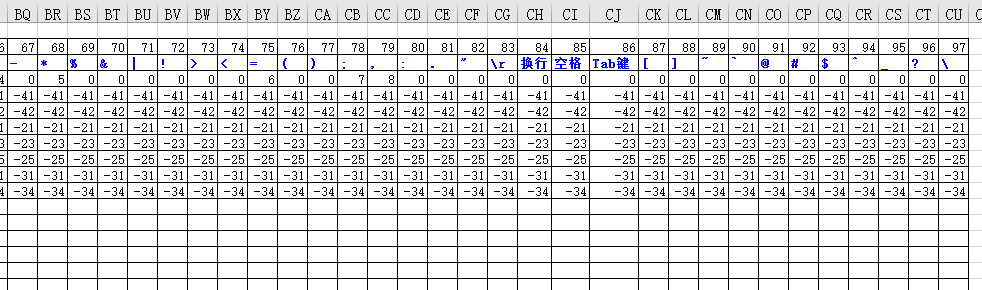


## 状态转换矩阵（三号+楷体+加粗）

此处请按照Small C语言的状态转换图修改设计文档模板中的“状态转换矩阵表”，并将该表的以图片的形式写于此处。







## 数据结构设计（三号+楷体+加粗）

此处请根据词法分析的代码讲解词法分析器的数据结构设计，主要包括：单词流的数据结构、源代码的存储结构、状态转换矩阵存储结构、常量信息表存储结构、变量信息表存储结构是如何定义的

//单词流的数据结构

typedef struct signDuality

{

int kind; //单词的类型

char value[20]; //单词的值

}\*pDualistic, Dualistic;

//源代码的存储结构

printf("源代码读入\n");

if ((fin=fopen("Test.c","r")) == NULL)

{

printf("Cannot open infile\n");

return 0;

}

//状态转换矩阵的存储结构

FILE \*fp;

row=0;col=0;

if((fp=fopen("zhjz.txt","r")) == NULL)

{

printf("文件不存在！");

exit(0);

}

//常量信息表存储结构

struct ConstInfo //常量信息表定义

{

int StoreType;//常量类型

char name[20]; //常量名字

int Value; //常量

}constTab[100];

//变量信息表存储结构

struct VarInfo//变量信息表定义

{

int StoreType; //变量类型

char content[STB\_MAX\_LEN]; //变量内容

}signTab[100];

## 函数设计（三号+楷体+加粗）

此处主要结合代码介绍词法分析函数wordAnalyse、状态处理函数Process的设计思路。

bool wordAnalyse(pDualistic pDu)

{

//定义字符数组Symbol[]用于存储所有的字符

char SymbolArr[]="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789/{}+-\*%&|!><=();,:.\"\r\n \t[]~`@#$^\_?\\";

k=0;

bool Tag=true;

iPos=0;//用于存储当前字符在源代码中的位序号

int iSymbol;//存储当前字符

//调用函数ReadToStaArray()，读入文件zhjz.txt（该文件存储的是状态转换矩阵）到内存StateTbl[][]中

ReadToStaArray();

int iState=1;//定义整型变量iState，并初始化为初始状态1

//循环读入源代码中的每一个字符，对单词进行提取

while(iPos<strlen(proBuffer)&&Tag) //如果当前字符的位序号i小于源代码的长度并且当前状态Tag为真

{

iSymbol=proBuffer[iPos++];

wordget[k++]=iSymbol; //将源代码中的第i个字符追加到wordget[]数组的末尾，wordget[]是用于存储正在提取的单词的缓冲区

row=iState;

col=SearchSymbol(SymbolArr,iSymbol);//定位当前字符在字符数组Symbol[]中的下标号,将这个下标号存入整型变量col中，用于记录状态转换矩阵的列号

iState=StateTbl[row-1][col];//从状态转换矩阵StateTbl[][]中查找iState状态在读入下标为col的字符时要转换成的状态

Tag=Process(&iState,pDu);

if(!Tag) return false;

}

return true;

}

bool Process(int \*iState,pDualistic ptmp)

{

if(\*iState==0)//如果当前状态为0，则说明单词拼写错误

return false;

else if(\*iState<0)//如果当前状态<0，则说明该单词已经提取完毕，需要根据状态的值判断该单词的类型

{

wordget[k-1]='\0';

iPos--;

if(\*iState==-1) //标识符

{

//将wordget[]中的单词存入单词流

ptmp=ptmp+ListLength;

ptmp->kind=1;

strcpy(ptmp->value,wordget);

//将标识符单词存入变量信息表

InsertId(ptmp->kind,ptmp->value);

ListLength++;

}

else if(\*iState>=-64&&\*iState<=-54) //运算符

{

//将wordget[]中的单词存入单词流

ptmp=ptmp+ListLength;

ptmp->kind=-(\*iState);

strcpy(ptmp->value,wordget);

ListLength++;

}

else if((\*iState)>=-84&&(\*iState)<=-70) //界符

{

//将wordget[]中的单词存入单词流

ptmp=ptmp+ListLength;

ptmp->kind=-(\*iState);

strcpy(ptmp->value,wordget);

ListLength++;

}

else if((\*iState)>=-87&&(\*iState)<=-85) //常量

{

//将wordget[]中的单词存入单词流

ptmp=ptmp+ListLength;

ptmp->kind=-(\*iState);

strcpy(ptmp->value,wordget);

//将单词存入常量信息表

InsertConst(wordget,atoi(ptmp->value));

ListLength++;

}

strcpy(wordget,"");

k=0;

\*iState=1;

}

else

row=\*iState;

return true;

}

# 语法分析器设计（标题1+三号+黑体+加粗+居中）

简要介绍利用预测分析法进行语法分析的大概思路。

验证这个词法分析生成的串可以由源语言的文法生成

## Small C语言的文法设计（三号+楷体+加粗）

正文。（正文采用宋体+小四+行距1.5倍）

|  |  |  |  |
| --- | --- | --- | --- |
| 产生式编号 | 产生式左部 | 产生式右部 | 代码 |
| 001 | 程序 | 语句部分 | 200 201 |
| 002 | 语句部分 | 声明部分 语句序列 | 201 202 207 |
| 003 | 声明部分 | 变量定义；声明部分 | 202 203 020 202 |
| 004 |  | ε | 202 000 |
| 005 | 变量定义 | 基本类型 标识符列表 | 203 206 204 |
| 006 | 标识符列表 | 标识符 标识符列表1 | 204 001 205 |
| 007 | 标识符列表1 | ，标识符 标识符列表1 | 205 019 001 205 |
| 008 |  | ε | 205 000 |
| 009 | 基本类型 | int | 206 002 |
| 010 | 语句序列 | 语句行 语句序列 | 207 208 207 |
| 011 |  | ε | 207 000 |
| 012 |  |  |  |
| 013 | 语句行 | 语句； | 208 209 020 |
| 014 | 语句 | 标识符 标识符起始语句 | 209 001 210 |
| 015 |  | ε | 209 000 |
| 016 | 标识符起始语句 | =表达式 | 210 032 211 |
| 017 | 表达式 | 操作数 运算符 操作数 | 211 212 213 |
| 018 |  | 操作数 | 211 212 |
| 019 | 操作数 | 标识符 | 212 001 |
| 020 |  | 常量 | 212 214 |
| 021 | 运算符 | + | 213 030 |
| 022 |  | \* | 213 031 |
| 023 | 常量 | 整型常量 | 214 045 |
| 024 |  |  |  |

## LL1分析表（三号+楷体+加粗）

**由于时间关系，我们组没时间做了，而且有好多细节要问，最终没做完。这个是老师的实例，无法复制文字，只能用图片了**

### 计算First集（四号+宋体+加粗）



### 计算Follow集（四号+宋体+加粗）



### 计算Select集（四号+宋体+加粗）



### LL1分析表（四号+宋体+加粗）

****



## 语法分析器数据结构设计（三号+楷体+加粗）

此处请根据语法分析的代码讲解语法分析器的数据结构设计，主要包括：文法的存储结构、LL1分析表的存储结构、非终结符的存储结构、终结符的存储结构、栈的存储结构是如何定义的

//文法的存储结构

int read\_wenfa();//从存储文法的文件wenfa.txt中读取文法到内存Vp\_array[][]数组中

//LL1分析表的存储结构

char \*LL1\_array[][16]={ //定义二维指针数组LL1\_array，每一个单元指向一个字符串，该字符串是LL1预测分析表表格中的产生式

{" "," "," ","047058059054201055"," "," "," "," "," "," "," "," "," "," "," "," "},

{"202205","202205","202205"," "," "," "," "," "," "," "," "," "," "," "," "," "},

{"000","212060202","212060202"," "," "," "," "," "," "," "," "," "," "," "," "," "},

{"001213"," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "},

{" ","045","046"," "," "," "," "," "," "," "," "," "," "," "," "," "},

{"001084209060211"," "," "," "," "," "," ","000"," "," "," "," "," "," "," "," "},

{"208207"," "," "," "," "," "," "," "," "," "," "," "," ","208207","208207"," "},

{" "," "," "," "," "," "," "," ","000"," ","210208","210208"," "," "," "," "},

{"001"," "," "," "," "," "," "," "," "," "," "," "," ","209"," "," "},

{" "," "," "," "," "," "," "," "," "," "," "," "," ","085","086"," "},

{" "," "," "," "," "," "," "," "," "," ","070","071"," "," "," "," "},

{"001084206060211"," "," "," "," "," "," ","000"," "," "," "," "," "," "," "," "},

{" ","204203","204203"," "," "," "," "," "," "," "," "," "," "," "," "," "},

{" "," "," "," "," "," "," "," ","000","061001213"," "," "," "," "," "," "},

};

//非终结符的存储结构

struct Vn //定义结构体表示非终结符

{

int kind; //非终结符编码

char name[50];//非终结符名字

};

//定义数组Vn\_array[],存储所有的非终结符,每一个单元存储非终结符的编码

//终结符的存储结构

struct Vt //定义结构体表示终结符

{

int kind; //终结符编码

char name[50];//终结符名字

};

//定义数组Vt\_array[],存储所有的终结符,每一个单元存储终结符的编码,因此这是个指针数组

//栈的存储结构

int stack[MAX]; //定义栈stack，栈中存储的是每一个终结符、非终结符的编码转换成的整数

int stacklength=0; //定义stacklength，用于存储栈的长度

int top; //top存储栈顶元素

## 语法分析器函数设计（三号+楷体+加粗）

此处主要结合代码介绍语法分析函数ll1\_analyzing、将非终结符定义的产生式压栈的函数ll1array\_push的设计思路。

void ll1array\_push(int cutchar)

//将LL1分析表中以栈顶元素top为行号、以单词流中的单词TokenList[i]为列号的表格中存储的产生式逆序压入栈stack中

{

int i,j;

i=Vn\_idx(); //读取栈顶元素top中的非终结符在数组Vn\_array[]中的下标号，返回该下标号并赋值给变量i

j=Vt\_idx(cutchar); //读取单词编码为cutchar的终结符在数组Vt\_array[]中的下标号，返回该下标号并赋值给变量j

strcpy(tp,LL1\_array[i][j]); //从LL1分析表中读取行号为i，列号为j的表格中存储的产生式右部，用指针tp指向该产生式的字符串

char \*p=tp ;

char tmp[3]="";

int code[20];

for(int n=0;n<strlen(tp)/3;n++)

{

for(int k=0;k<3;k++) //取出tp指针指向的字符串中的3位存入数组tmp[]中

tmp[k]=\*p++;

tmp[k]='\0';

code[n]=atoi(tmp); //将编码字符串转换成整数存入code[]数组中

// printf("%s ",Name(code[n]));

}

reve(code,n); //将tp指向的产生式字符串逆序

pop(); //栈顶元素top出栈

for(k=0;k<n;k++)

push(code[k]); //将tmp[]中的字符串转换成整数压入栈stack中

// printf("=>");

if (top == 0) //如果栈顶元素top为0,说明栈顶元素是空串

pop(); //将栈顶的空串出栈

}

# 主函数设计（三号+黑体+加粗+居中）

结合main()函数代码，介绍主函数的设计思路

int main(int argc, char\* argv[])

{

pDualistic ptmp = TokenList;

printf("源代码读入\n");

read\_SouceCode(); //从文件test.c读入源代码到数组proBuffer[]中

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n源代码读入成功，源代码如下:\n%s",proBuffer);

printf("\n按任意键继续\n"); getch();

printf("\n预处理\n");

pretreatment(); //对源代码进行预处理，去掉源代码中的注释语句

printf("\n词法分析\n");

wordAnalyse(ptmp);//词法分析，将所有的单词提取出来并存入单词流TokenList【】

OutPut\_Result(); //将单词流TokenList[]中的所有单词写入文件result.txt中

printf("写回常量表和标识符表\n");

OutPut\_Const(); //将常量信息表constTab[]中的所有常量写入文件Const.txt中

OutPut\_Sign(); //将变量信息表signTab[]中的所有变量写入文件Sign.txt中

printf("\n写入完毕\n按任意键继续\n");

getch();

printf("\n词法分析\n");

read\_wenfa(); //从存储文法的文件wenfa.txt中读取文法到内存Vp\_array[][]数组中

init\_stack(); //初始化栈stack

ll1\_analyzing(); //用LL1分析法进行语法分析

return 0;

}

# 测试（三号+黑体+加粗+居中）

给出2个测试用例，并给出测试结果，测试结果可以是截图的形式。

可参考如下例子列举测试用例

测试用例1（符合Small C语法的例子）：

main()

{

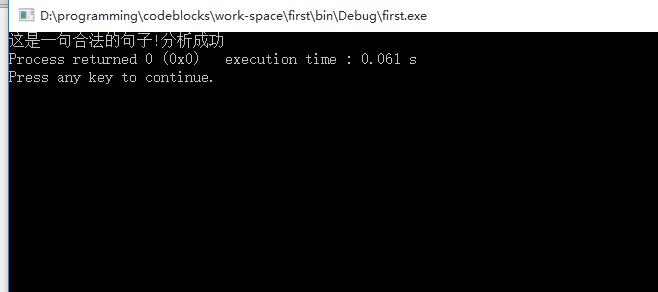
int a;

a=3;

/\*wowoow\*/

}

测试结果：



测试用例2（不符合Small C语法的例子）：

main()

{

int a,b;

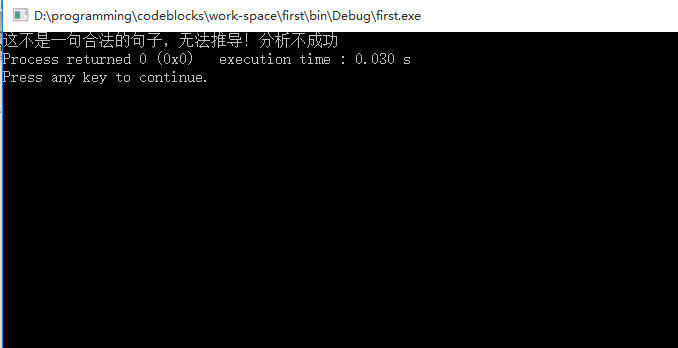
a=3

b=a+2;

/\*wowoow\*/

}

测试结果：



分析推导不成功的原因，上面的源代码有什么语法错误。

缺少分号